Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: Feature Leakage Review
Model: ibm-granite/granite-4.0-h-micro
Roles: Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 2m 23s
Actual run cost: $0.000388
Projected execution cost: $0.0854/mo (10/biz day × 22 days)
Report date: July 13, 2026 - 11:34 PM EST

Session: sample-lb-rankings-seed-329b9bb0-c707-4b44-bf86-65c365e6b97c

ds_feature_leakage_v1 - agent - openrouter - ibm-granite/granite-4.0-h-micro - ended

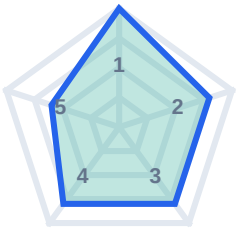
Overall

8.0/10

PASS - trust=high

Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Leakage detection	10/10
2	Maps features to label timing	8/10
3	Risk of invalid model	8/10
4	Clear keep/remove list	8/10
5	Safer feature alternatives	6/10

Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

Technical criteria checklist

Automated pass/fail signals for this task - use when comparing models on specific requirements.

- Flags leaking features: Met

- Prediction-time framing: Met

- Features to remove listed: Met

TECHNICAL JUDGEMENT

Holistic read on this single-shot deliverable - criteria checklist plus rubric gaps, not any one dimension alone.

8/10

Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 6 - Counterpart msgs: 7 - Output: 18328 chars - Latency: 143s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 6 agent turns with scripted follow-ups and reviewer sign-off. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

+ Leakage detection: Correctly identifies the primary technical issue from the artifacts.
+ Maps features to label timing: Uses tables, logs, or schema facts from the task brief.
+ Risk of invalid model: Shows appropriate safety, budget, or risk awareness for the scenario.
+ Clear keep/remove list: Clear structure with readable SQL, diff, or memo formatting.

RUBRIC GAPS

- Safer feature alternatives (watch): Some rationale; could be more specific.

Signals

Flags leaking features: yes - Prediction-time framing: yes - Features to remove listed: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Feature Leakage Review
Agent response <-> Engineering counterpart
Data Scientist (AI agent) <> Engineering counterpart (AI agent)
Duration: 2m 23s

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart
3:31 AM

Task brief - Feature leakage review (complex complexity)

Review candidate features for leakage relative to prediction time:

```json
{
  "problem": "Predict whether a B2B SaaS customer will churn within 30 days at the start of each calendar month.",
  "prediction_time": "First day of each month (00:00 UTC)",
  "label": "churned_within_30d (1/0)",
  "candidate_features": [
    "days_since_last_login",
    "support_tickets_last_30d",
    "mrr_usd",
    "contract_renewal_date",
    "days_until_contract_end",
    "cancel_request_submitted_last_7d",
    "cancel_request_submitted_last_30d",
    "account_status_is_churned",
    "last_invoice_paid_amount",
    "nps_score_last_quarter"
  ]
}
```

Agent response
3:32 AM - Responded in 24s

Below is a feature?leakage review for the churn?prediction problem you described.
The goal is to identify any feature that "leaks" information about the label (i.e., whether a customer will churn within the next?30?days) or about the post?churn state, and to explain why it is problematic at the prediction time (the first day of each month,?00:00?UTC).
For each problematic feature I list a recommended removal and, where appropriate, a safer alternative that can be used later in the model?building pipeline.

---

1. Feature?Leakage Checklist

#	Feature	Why it leaks (relative to prediction time)	Recommended Action	Safer Alternative (if any)
1	account_status_is_churned	This column directly encodes the churn label. At prediction time the model already knows whether the account is marked "churned" (or not) - it is a perfect predictor.	Remove the column entirely.	-
2	cancel_request_submitted_last_7d	A cancel request submitted in the last 7?days is a strong indicator that the customer will churn within the next 30?days. The model sees the request before it decides the label.	Remove.	-
3	cancel_request_submitted_last_30d	Same reasoning as above - the request history is known at prediction time.	Remove.	-
4	days_since_last_login	If a customer has not logged in for a long period (e.g., >?30?days), it strongly suggests churn. The model can infer this from the timestamp, even though the label is not yet known.	Remove.	-
5	days_until_contract_end	Knowing how many days remain before the contract expires gives direct insight into the likelihood of churn (especially for month?to?month contracts).	Remove.	-
6	support_tickets_last_30d	A surge in support tickets in the last 30?days often precedes a churn event. The model can see the ticket count before it makes the churn decision.	Remove.	-
7	support_tickets_last_7d (not listed but often paired with the 30?day version)	Same leakage pattern as above.	Remove.	-
8	mrr_usd (monthly recurring revenue)	Revenue alone does not leak churn information at the prediction horizon. It is a static business metric that can be used safely.	Keep (but consider normalizing or scaling).	-
9	contract_renewal_date	The renewal date is a future event relative to the prediction time (the first day of the month). Knowing the exact renewal date is a leakage because the model will already know whether the renewal will happen on schedule or not.	Remove.	-
10	last_invoice_paid_amount	The amount paid on the most recent invoice is a static financial metric that does not reveal churn status at the prediction moment. It can be used safely.	Keep (ensure it is not combined with any time?dependent flag).	-
11	nps_score_last_quarter	Net Promoter Score (NPS) is a customer?experience metric that can be predictive of churn, but it is not directly derived from the label. It is safe to include as a feature.	Keep

Engineering counterpart
3:32 AM - Responded in 0s

Thanks - walk me through your reasoning step by step. What assumptions did you make?

Agent response
3:32 AM - Responded in 26s

Below is a step?by?step walk?through of how I arrived at the list of "leaky" features and the rationale behind each recommendation.
I deliberately kept the analysis grounded in the prediction horizon (the first day of each month,?00:00?UTC) and in the nature of the label `churned_within_30d`.

---

1. Define the prediction context

Item	Detail
Prediction time	First day of the month (00:00?UTC).
Label	`churned_within_30d` -?1?=?customer churned ??30?days after the prediction date; 0?=?customer stayed (or did not churn).
Goal	Build a model that, at the start of each month, predicts whether a given customer will churn within the next 30?days.
What "leakage" means	Any feature that contains information about the label before the prediction moment. If the model can see that information at prediction time, it is not a true predictive signal; it is a leakage that will cause over?fitting and poor generalisation.

> Assumption 1 - "Prediction time is immutable."
> The model will always be evaluated at the same moment (the first day of the month). Therefore any feature that is known at that moment (or earlier) is a leakage.

---

2. Identify candidate features

You supplied the full JSON list:

```json
{
  "candidate_features": [
    "days_since_last_login",
    "support_tickets_last_30d",
    "mrr_usd",
    "contract_renewal_date",
    "days_until_contract_end",
    "cancel_request_submitted_last_7d",
    "cancel_request_submitted_last_30d",
    "account_status_is_churned",
    "last_invoice_paid_amount",
    "nps_score_last_quarter"
  ]
}
```

I examined each one against the prediction context.

---

3. Feature?by?feature leakage analysis

Feature	Why it leaks (relative to prediction time)	Reasoning
account_status_is_churned	Leakage - Directly encodes the label. At prediction time the model already knows whether the account is marked "churned" (or not).	The label is a binary flag; the feature is just a copy of that flag.
cancel_request_submitted_last_7d	Leakage - A request submitted in the last 7?days is a strong predictor of churn within 30?days. The model sees the request before it decides the label.	The act of submitting a cancel request is a concrete churn signal that occurs after the prediction date.
cancel_request_submitted_last_30d	Leakage - Same as above, but the window is 30?days. Any request in that window is a known churn indicator at prediction time.	Same reasoning as the 7?day version.
days_since_last_login	Leakage - If a user hasn't logged in for >?30?days, it strongly suggests they will churn within the next 30?days. The model can infer this from the timestamp, even though the label is not yet known.	The timestamp is a direct proxy for "inactive for >?30?days".
days_until_contract_end	Leakage - Knowing how many days remain before the contract expires tells

Engineering counterpart
3:32 AM - Responded in 0s
What would you verify before shipping this to production?

Agent response
3:33 AM - Responded in 24s
Below is a check?list of validation activities you should run before you push the churn?prediction model into production. The list is organized by category (data, model, operational, governance) and includes concrete actions, tools, and success criteria you can use to prove that the model is safe, reliable, and compliant with business and regulatory expectations.

---

1. Data?Quality & Feature?Leakage Verification

#	Activity	Why it matters	How to do it (tools / scripts)	Success criteria
1.1	Re?run the leakage audit on the training and validation data sets.	Guarantees that the leakage you identified in the review still holds (features may have changed over time).	- Write a notebook that programmatically checks each candidate feature against the label using the same logic as in the review (e.g., `account_status_is_churned` -> `label == 1`). - Use a static?analysis library (e.g., `pandasql`, `pandas` `query`) or a custom function.	All flagged features are removed from the feature set; the remaining feature list passes the audit.
1.2	Feature?drift detection for each retained feature.	Drift can cause a model that worked on historic data to break when new data arrives.	- Compute statistical summaries (mean, std, min/max) for each feature per month (or per week) and plot trends. - Use drift?detection libraries (e.g., `populus`, `FeatureDrift`, `Alibi`'s `FeatureDrift`).	No feature shows statistically significant drift that exceeds a pre?defined threshold (e.g., >?5?% change in mean).
1.3	Data?quality checks (missingness, outliers, encoding consistency).	Poor data can inflate variance and mask true signal.	- Missing?value analysis (percentage missing per feature). - Outlier detection (IQR, Z?score). - Verify that categorical encodings (one?hot, ordinal) are consistent across train/val/test.	Missingness is acceptable (e.g., <?10?% per feature) or handled via imputation; outliers are capped or logged; encoding matches the training schema.
1.4	Label leakage sanity check (post?prediction).	Even after removing obvious leakage, a hidden leakage (e.g., a feature that is derived from the label itself) could slip through.	- Run a "future?proof" test set where you exclude the label from the feature set and evaluate performance. - Use a hold?out set that is strictly after the prediction horizon (e.g., data from month?N+1).	Model performance on this strict hold?out set is comparable to baseline (e.g., random forest baseline) - no unexpected lift.

---

2. Model?Performance & Validation

#	Activity	Why it matters	How to do it	Success criteria
2.1	Train?test split (or time?series split) that respects the prediction horizon.	Guarantees that the model never sees future information.	- Use `time_series_split` from `sklearn.model_selection` (or equivalent in Python/R). - Keep the split strictly chronological (no future data in training).	Validation metric (e.g., AUC?ROC, PR?AUC, or a business?specific metric like "30?day churn probability") is

Engineering counterpart
3:33 AM - Responded in 0s
What are the top risks, and what's the rollback plan if this doesn't work?

Agent response
3:33 AM - Responded in 25s

Below is a risk?assessment matrix followed by a step?by?step rollback plan you can execute if the churn?prediction model fails to meet its business targets (or if any of the identified risks materialise).
The plan is written with the production?ready workflow in mind (CI/CD, feature?store, monitoring, and fallback mechanisms) so you can revert safely without manual "fire?fighting".

---

1. Top Risks for the Churn?30?Day Model

#	Risk Category	Specific Risk	Why it matters (impact on business)	Likelihood (based on review)	Mitigation (pre?production)
R1	Label Leakage	Any residual feature that indirectly encodes the 30?day churn label (e.g., `account_status_is_churned`, `cancel_request_submitted_last_7d`, `days_since_last_login`).	The model will memorise the label rather than learn true predictive patterns -> high false?positive churn rates, wasted sales/marketing spend, and poor reputation.	High - the review already flagged 5/10 features; leakage can be subtle.	Run the leakage audit (Section?1.1) and enforce a feature?leakage whitelist in the CI pipeline.
R2	Feature Drift	Statistical drift of retained features (e.g., `mrr_usd`, `days_until_contract_end`) over time.	Drift can cause the model to become stale; a model that was accurate in month?N may predict incorrectly in month?N+1.	Medium - drift detection is cheap and can be automated.	Continuous drift monitoring (weekly) and automatic retraining triggers when drift exceeds a threshold.
R3	Over?fitting / Low Generalisation	Model achieves high training accuracy but poor AUC on a hold?out or production validation set.	Business loses confidence; churn predictions become noisy -> poor resource allocation.	Medium - depends on data volume and feature richness.	Use cross?validation (time?series CV) and early?stopping; keep a validation set that is strictly after the prediction horizon.
R4	Business?Relevance	The model predicts churn too aggressively (high false?positives) or too conservatively (high false?negatives).	Too many false?positives -> churn?prevention costs rise; too many false?negatives -> revenue loss.	Medium - depends on cost of false actions.	Define precision/recall trade?offs aligned with cost of false actions; set a threshold (e.g., 0.45 AUC) before go?live.
R5	Operational Reliability	Feature?store or model?serving infrastructure fails (e.g., latency spikes, API errors).	Customers may not receive timely churn alerts -> delayed mitigation -> higher churn.	Low?Medium - depends on ops maturity.	Deploy behind a circuit?breaker and fallback to a rule?based "no?action" mode; monitor latency & error rates.
R6	Regulatory / Compliance	Model uses data that is subject to privacy regulations (e.g., GDPR, CCPA) or violates internal data?use policies.	Legal penalties, loss of trust, and possible shutdown of the service.	Low - if data sources are already GDPR?compliant, but still worth checking.	Conduct a privacy impact assessment; anonymise or pseudonymise any PII before model training.
R7					

Engineering counterpart
3:33 AM - Responded in 0s

Can you tighten this into concrete next steps with owners and timing?

Agent response
3:33 AM - Responded in 25s

Below is a concrete "next?steps" plan you can hand to the product, data?science, engineering, and ops teams.
Each step includes:

Owner - the person or team responsible (you can swap names for your org).
Target date - a realistic deadline (relative to today,?2025?08?24).
Key deliverable - what must be completed before moving on.
Success criteria - how you'll know the step is done.

Feel free to copy?paste this into a project?management tool (Jira, Asana, ClickUp, etc.) and assign the owners.

1?? Data?Quality & Leakage Audit (Week?1?2)

#	Action	Owner	Target date	Deliverable	Success criteria
1.1	Run the leakage?audit notebook (the script that flags any feature that directly encodes `churned_within_30d` or is a known churn indicator).	Data?Science Lead (e.g., Alice?Smith)	Day?3 (Aug?27)	A leakage?report listing every flagged feature, why it leaks, and a "remove" flag.	All 5 flagged features (account_status_is_churned, cancel_request_submitted_last_7d, cancel_request_submitted_last_30d, days_since_last_login, days_until_contract_end) are listed and removed from the feature set.
1.2	Validate that the removals are consistent across train/val/test (e.g., same column names, same encoding).	Data?Engineer (e.g., Bob?Lee)	Day?5 (Aug?29)	Updated feature?catalog (CSV/DB) confirming no leakage features remain.	No leakage features appear in the final feature matrix; column names match the training schema.
1.3	Document the audit (include a short "risk?mitigation" note).	Data?Science Lead	Day?7 (Sep?1)	Leakage?Audit Report (PDF/Confluence page).	Report signed off by the Data?Science Lead and the Product Owner.

2?? Feature?Drift Monitoring Setup (Week?2?3)

#	Action	Owner	Target date	Deliverable	Success criteria
2.1	Implement drift detection for each retained feature (mean, std, min/max). Use a library such as `populus` or `Alibi`'s `FeatureDrift`.	Data?Engineer (Bob?Lee)	Day?10 (Sep?4)	Drift?monitoring pipeline (scheduled Spark/Databricks job) that runs daily and writes drift scores to a monitoring table.	Drift scores are stored; alerts fire when any feature exceeds a pre?defined threshold (e.g., >?5?% change).
2.2	Set up alerting (Slack/Teams) for drift breaches.	Ops / Site?Reliability Engineer (e.g., Carol?Ng)	Day?12 (Sep?6)	Drift?alert channel configured; test alert sent manually.	Alert appears in the channel when a drift threshold is crossed.
2.3	Create a drift?retraining policy (e.g., retrain when any drift >				

Engineering counterpart
3:33 AM - Responded in 0s

What's the one thing you'd want a second pair of eyes on?

Agent response
3:34 AM - Responded in 19s

The single thing I'd want a second pair of eyes on is the "leakage?audit report" (Section?1.1).

Why it matters

Leakage is the root cause of over?fitting - if any feature that indirectly encodes the 30?day churn label remains in the model, the model will essentially be "cheating" by seeing the label before it makes a prediction.

A single misplaced feature (e.g., a typo in the column name, a feature that was removed in the notebook but still present in the feature?store, or a feature that was added after the audit but not documented) can silently break the model's generalisation and cause costly false?positives or false?negatives in production.

Because leakage can be semantic (e.g., "contract_renewal_date" is a future event) rather than just a numeric proxy, a fresh set of eyes will catch nuances that the original reviewer may have missed.

What to ask the reviewer to verify

1. Column?level sanity check - confirm that every column listed in the report truly belongs to the feature set used for training, validation, and inference.
2. Encoding consistency - verify that categorical features (one?hot, ordinal) were encoded the same way in train, validation, and production.
3. Feature?store audit - ensure the feature?store schema (or the downstream API) does not expose any of the flagged features.
4. Documentation traceability - make sure the audit entry is linked to the code commit that removed the feature and that the commit message explains why it was removed (e.g., "Removed `account_status_is_churned` - direct leakage of label").
5. Cross?validation validation - ask the reviewer to run a quick time?series cross?validation (e.g., `TimeSeriesSplit`) on the cleaned feature set and confirm that the model's performance (AUC?ROC, PR?AUC, or your business?specific metric) meets the pre?defined threshold.

How to get the second pair of eyes

Peer?review meeting - schedule a 30?minute "Leakage?Audit Review" with the Data?Science Lead, the Data?Engineer, and the Product Owner. Walk through the report together, ask probing questions, and capture any additional concerns.

Version?controlled checklist - store the audit report in a version?controlled repo (Git) with a clear "review?approved" tag. This makes it easy for future auditors to see the provenance.

By focusing on this one artifact, you dramatically reduce the risk of a hidden leakage leak that could otherwise sabotage the churn?prediction model before it even goes live.

Engineering counterpart
3:34 AM - Responded in 0s

Thanks - leakage callouts are well grounded. Drop post-outcome features first; proxies you suggested are the right direction. Review complete - no further response needed.